Expectation Maximization & Regression
Lecture 21

David Sontag
New York University

Slides adapted from Carlos Guestrin, Dan Klein, Luke Zettlemoyer, Dan Weld, Vibhav Gogate, and Andrew Moore

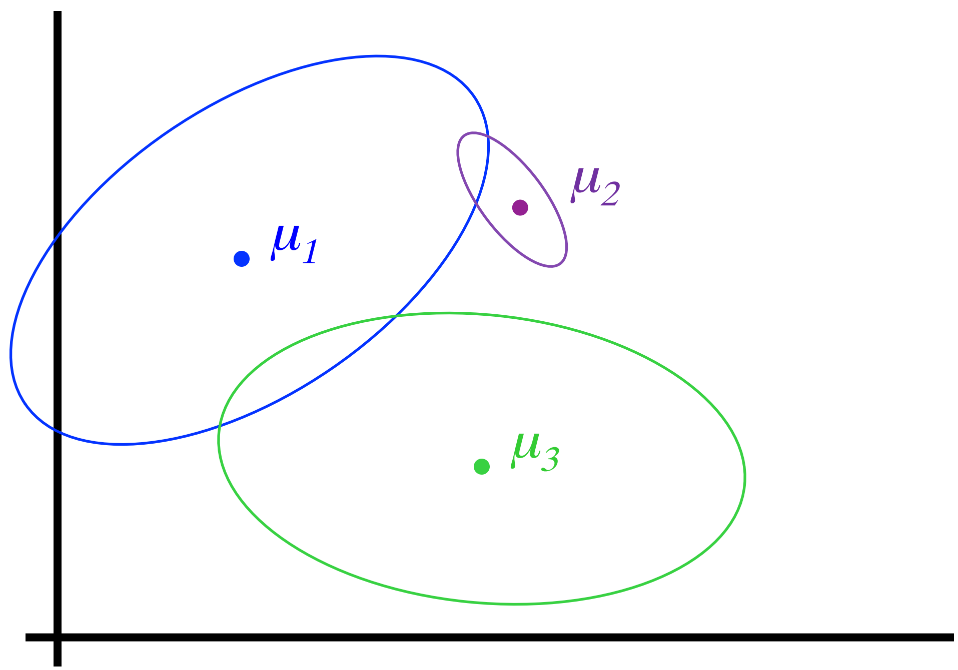
The General GMM assumption

- P(Y): There are k components
- P(X|Y): Each component generates data from a **multivariate Gaussian** with mean μ_i and covariance matrix Σ_i

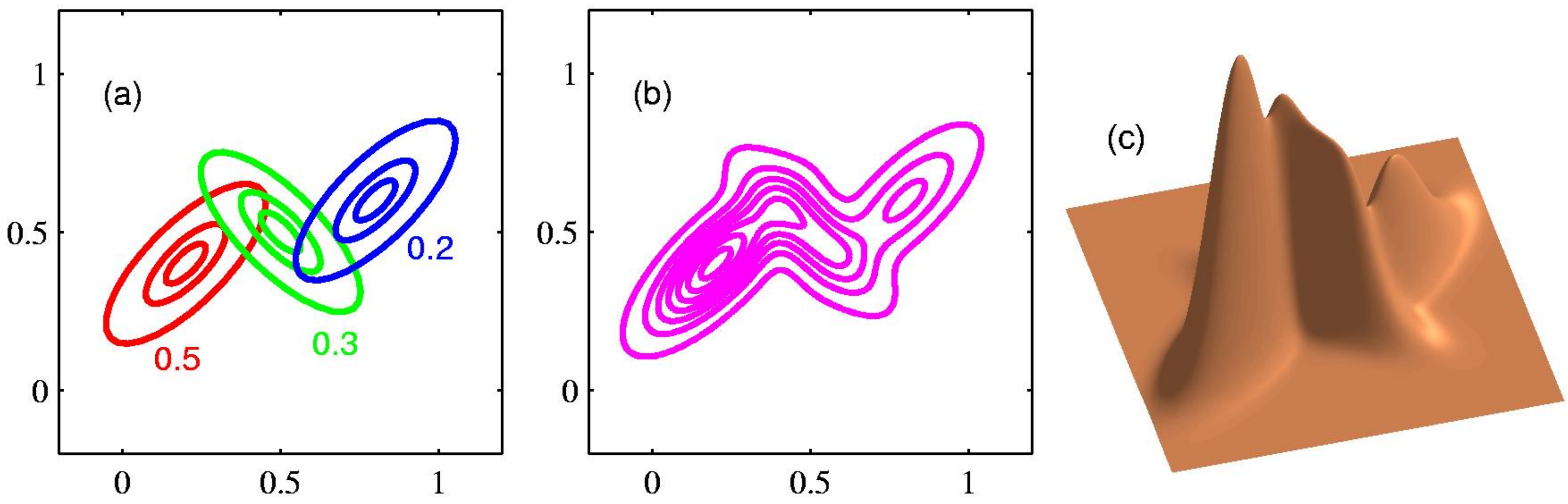
Each data point is sampled from a ***generative process*:**

1. Choose component i with probability $P(y=i)$
2. Generate datapoint ~ $N(m_i, \Sigma_i)$

Gaussian mixture model (GMM)



Mixtures of Gaussians



E.M. for General GMMs

$p_k^{(t)}$ is shorthand for estimate of $P(y=k)$ on t'th iteration

Iterate: On the *t*'th iteration let our estimates be

$$\lambda_t = \{ \mu_1^{(t)}, \mu_2^{(t)} \ldots \mu_K^{(t)}, \Sigma_1^{(t)}, \Sigma_2^{(t)} \ldots \Sigma_K^{(t)}, p_1^{(t)}, p_2^{(t)} \ldots p_K^{(t)} \}$$

E-step

Compute "expected" classes of all datapoints for each class

$$\mathrm{P}\left(Y_j = k \middle| x_j, \lambda_t\right) \propto p_k^{(t)} \mathrm{p}\left(x_j \middle| \mu_k^{(t)}, \Sigma_k^{(t)}\right)$$

Just evaluate a Gaussian at x_j

M-step

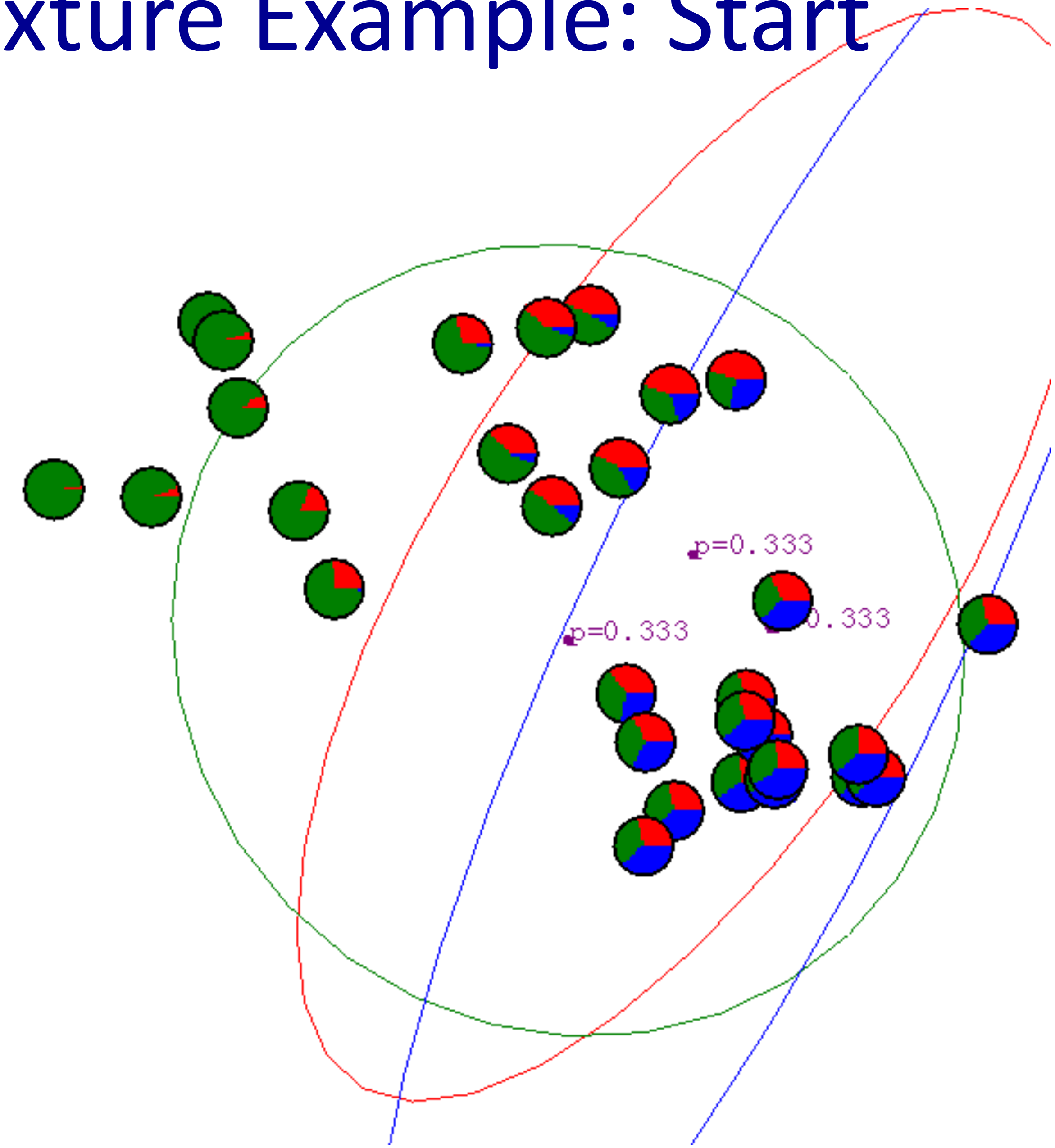
Compute weighted MLE for **μ** given expected classes above

$$\mu_k^{(t+1)} = \frac{\sum_j \mathrm{P}\left(Y_j = k \middle| x_j, \lambda_t\right) x_j}{\sum_j \mathrm{P}\left(Y_j = k \middle| x_j, \lambda_t\right)} \qquad \Sigma_k^{(t+1)} = \frac{\sum_j \mathrm{P}\left(Y_j = k \middle| x_j, \lambda_t\right) \left[x_j - \mu_k^{(t+1)}\right]\left[x_j - \mu_k^{(t+1)}\right]^T}{\sum_j \mathrm{P}\left(Y_j = k \middle| x_j, \lambda_t\right)}$$

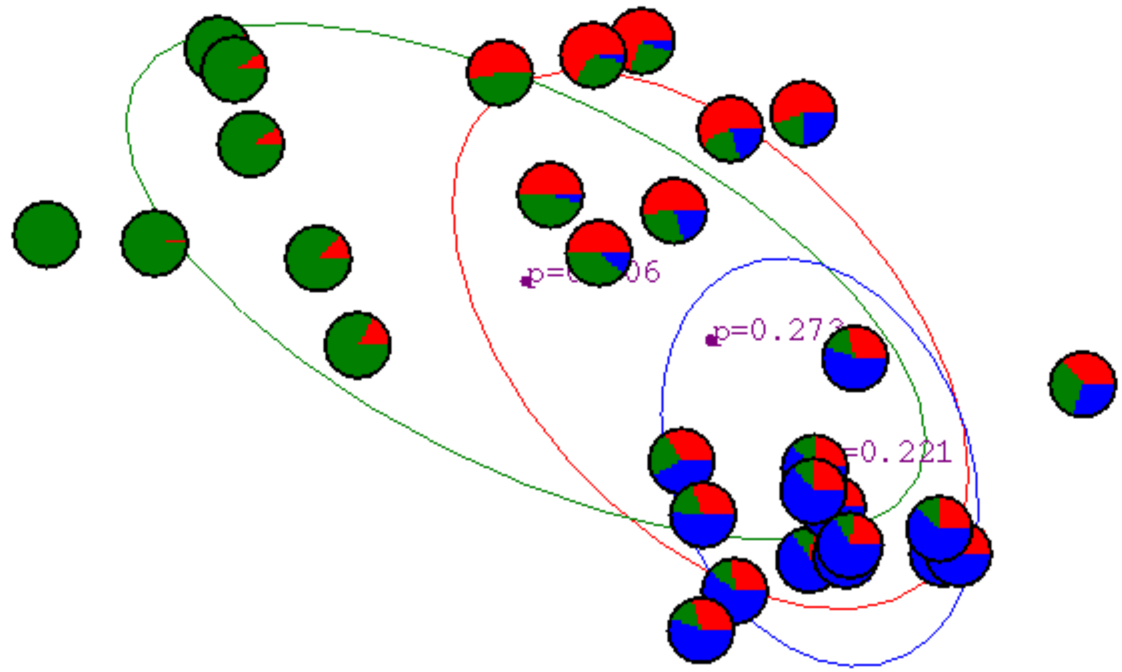
$$p_k^{(t+1)} = \frac{\sum_j \mathrm{P}\left(Y_j = k \middle| x_j, \lambda_t\right)}{m}$$

m = #training examples

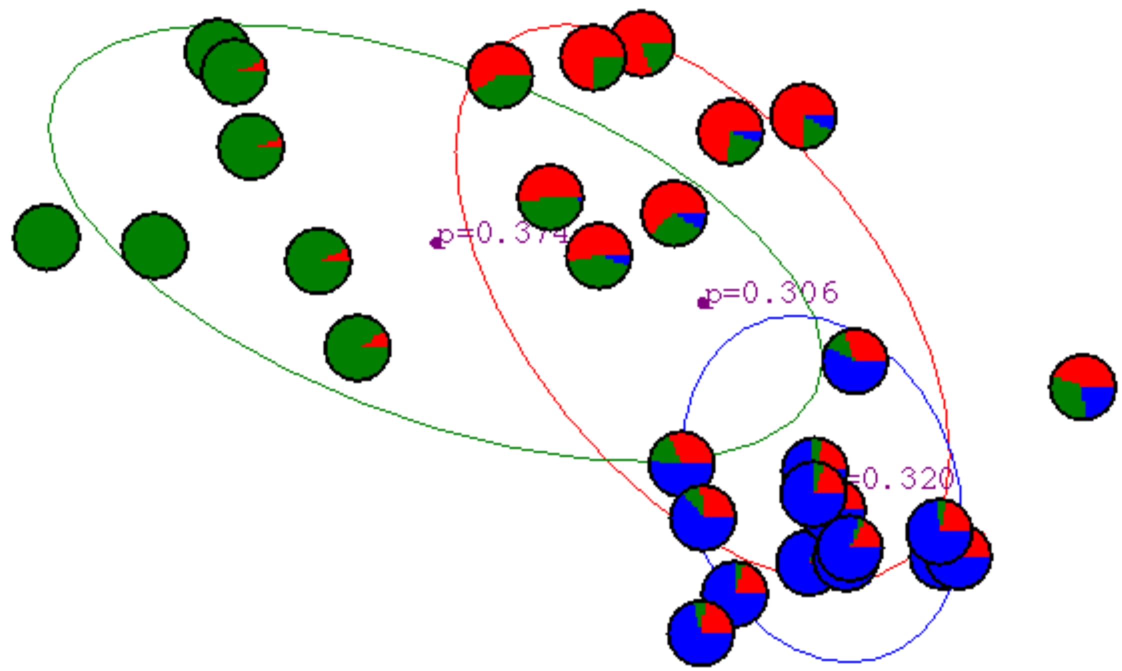
Gaussian Mixture Example: Start



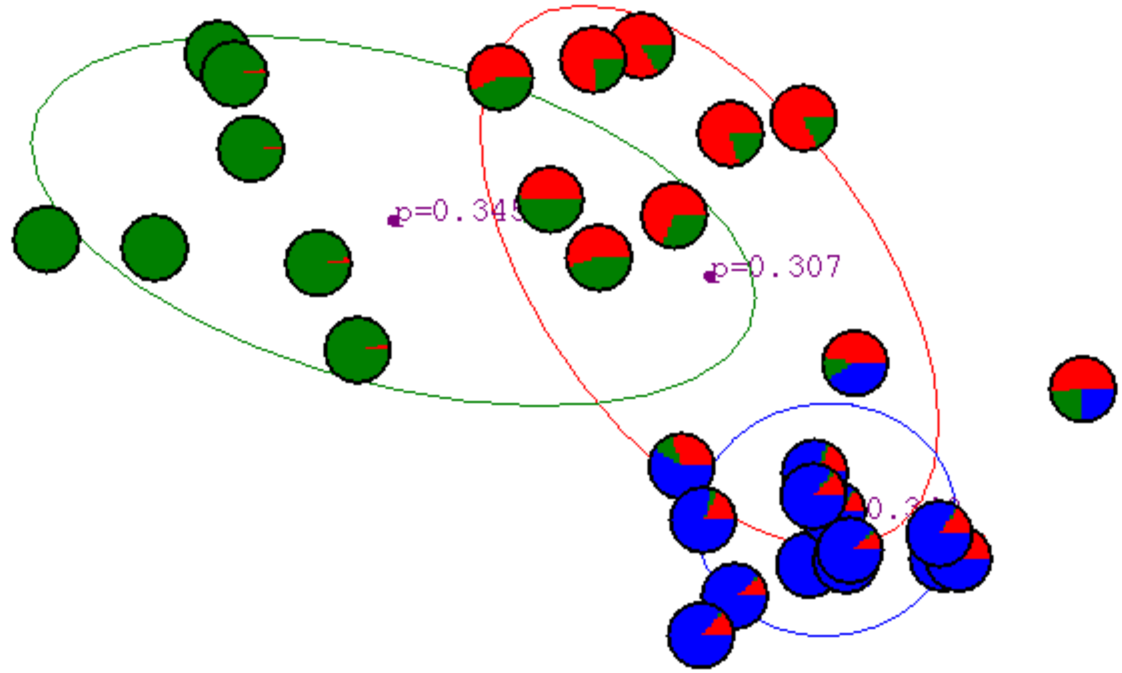
After first iteration



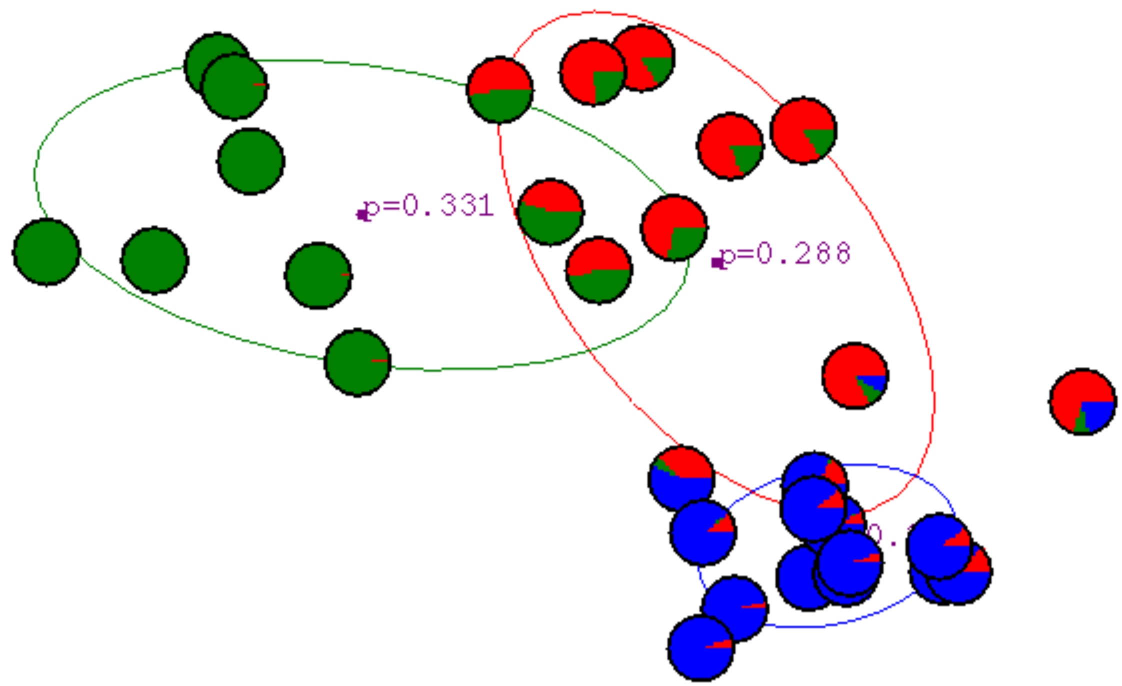
After 2nd iteration



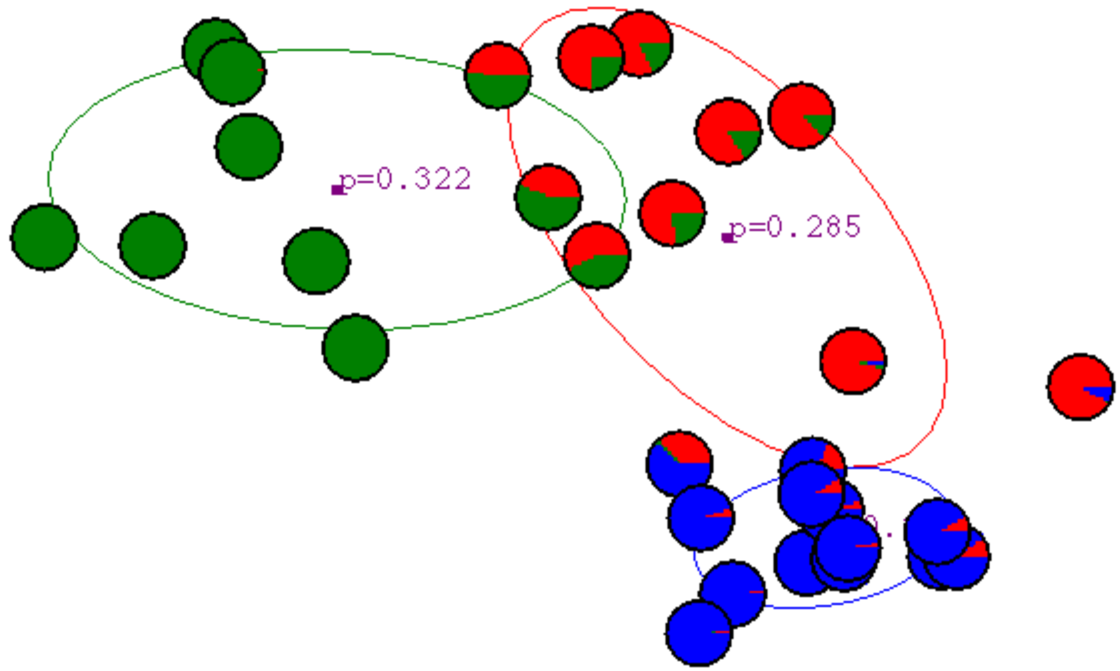
After 3rd iteration



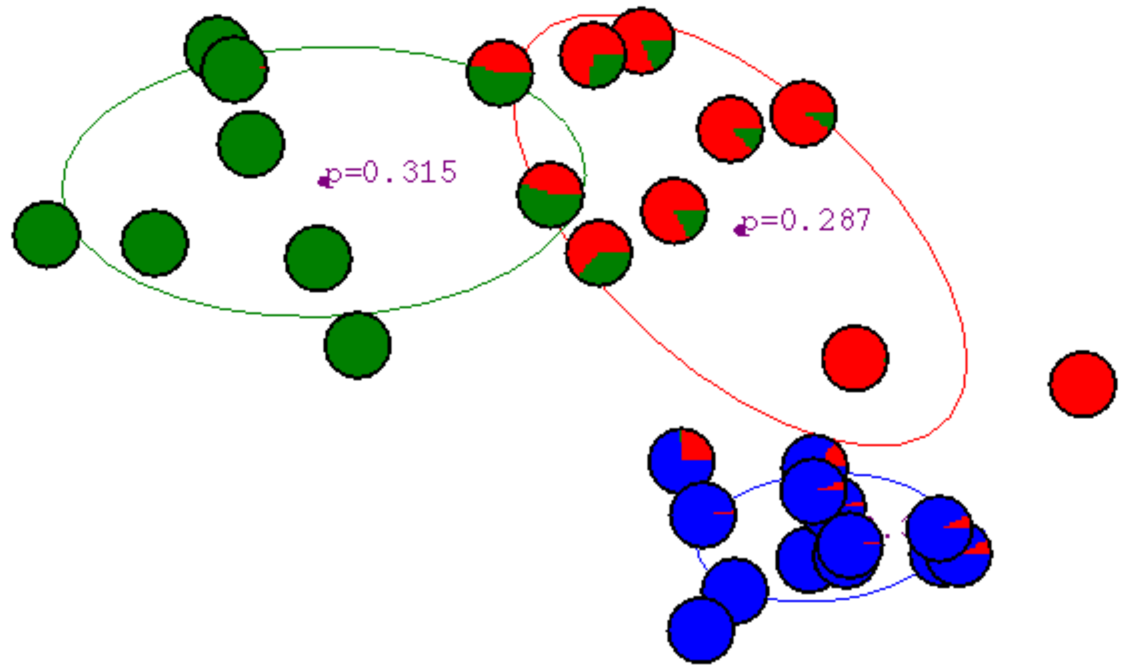
After 4th iteration



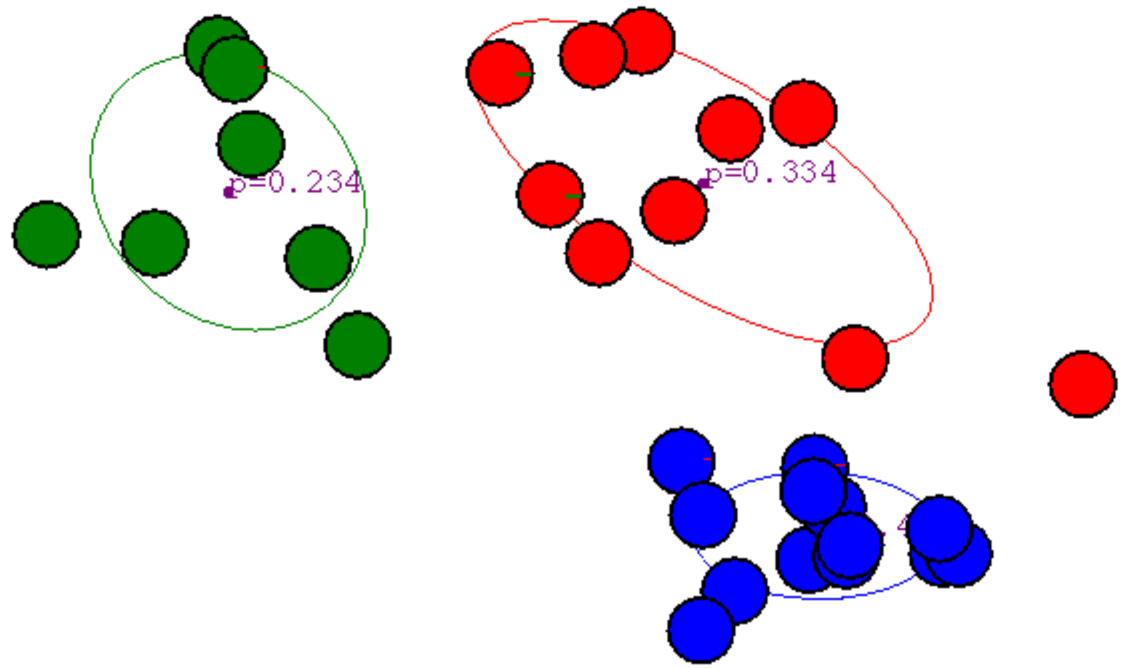
After 5th iteration



After 6th iteration



After 20th iteration



What if we do hard assignments?

Iterate: On the *t*'th iteration let our estimates be

$$\lambda_t = \{ \mu_1^{(t)}, \mu_2^{(t)} \ldots \mu_K^{(t)} \}$$

E-step

Compute "expected" classes of all datapoints

$$P\left(Y_j = k \middle| x_j, \mu_1 \ldots \mu_K\right) \propto \exp\left(-\frac{1}{2\sigma^2}\left\|x_j - \mu_k\right\|^2\right) \mathrm{P}\left(Y_j = k\right)$$

M-step

Compute most likely new **μ**s given class expectations

$$\mu_k = \frac{\sum_{j=1}^{m} P\left(Y_j = k \middle| x_j\right) x_j}{\sum_{j=1}^{m} P\left(Y_j = k \middle| x_j\right)}$$

$$\mu_k = \frac{\delta\left(Y_j = k, x_j\right) x_j}{\sum_{j=1}^{m} \delta\left(Y_j = k, x_j\right)}$$

δ represents hard assignment to "most likely" or nearest cluster

Equivalent to k-means clustering algorithm!!!

The general learning problem with missing data

- Marginal likelihood: **X** is observed,

  Z (e.g. the class labels **Y**) is missing:

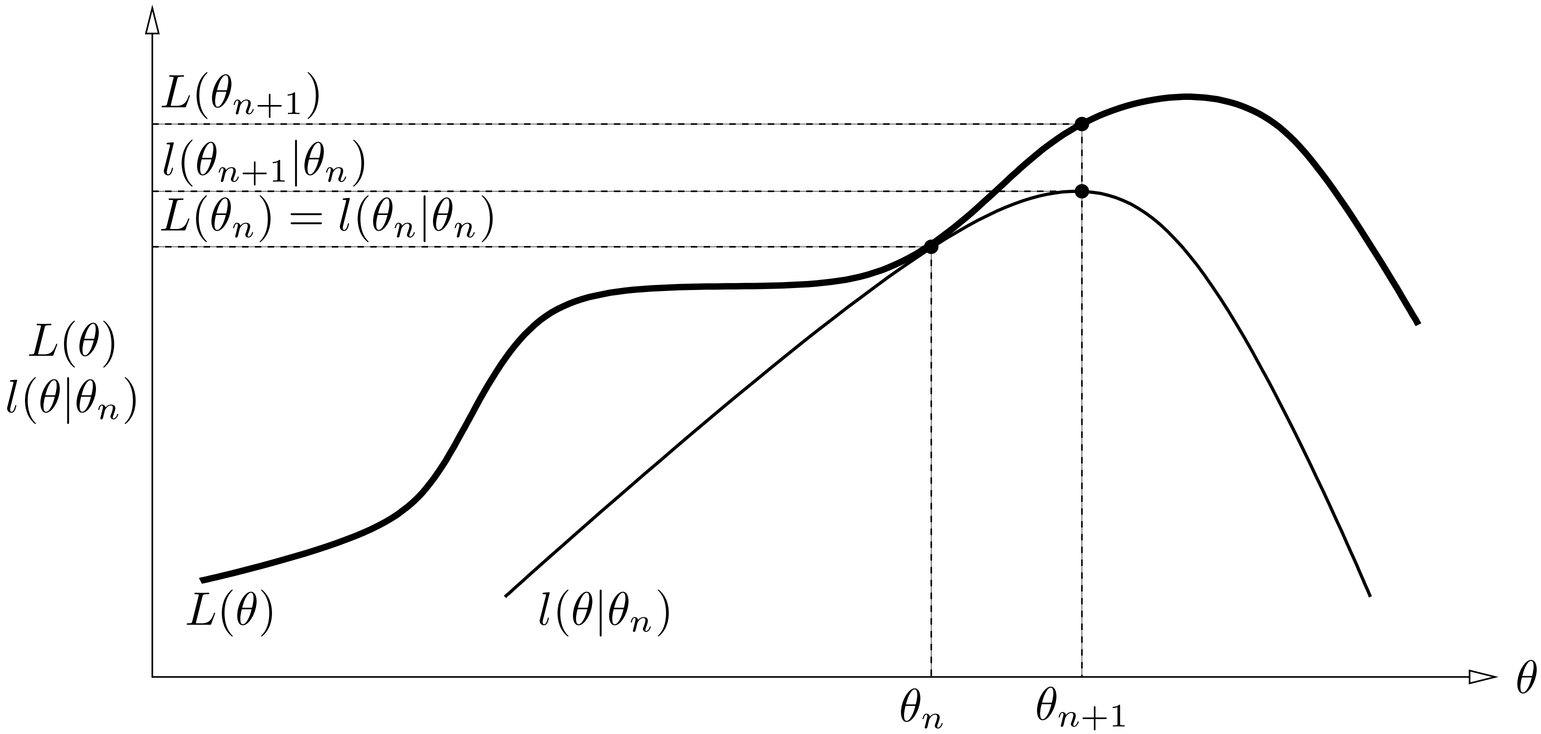
$$
\begin{aligned}
\ell(\theta : \mathcal{D}) &= \log \prod_{j=1}^{m} P(\mathbf{x}_j \mid \theta) \\
&= \sum_{j=1}^{m} \log P(\mathbf{x}_j \mid \theta) \\
&= \sum_{j=1}^{m} \log \sum_{\mathbf{z}} P(\mathbf{x}_j, \mathbf{z} \mid \theta)
\end{aligned}
$$

- Objective: Find argmax_θ $l(\theta\text{:Data})$

Properties of EM

- We will prove that
  - EM converges to a local maxima
  - Each iteration improves the log-likelihood
- How? (Same as k-means)
  - E-step can never decrease likelihood
  - M-step can never decrease likelihood

EM pictorially



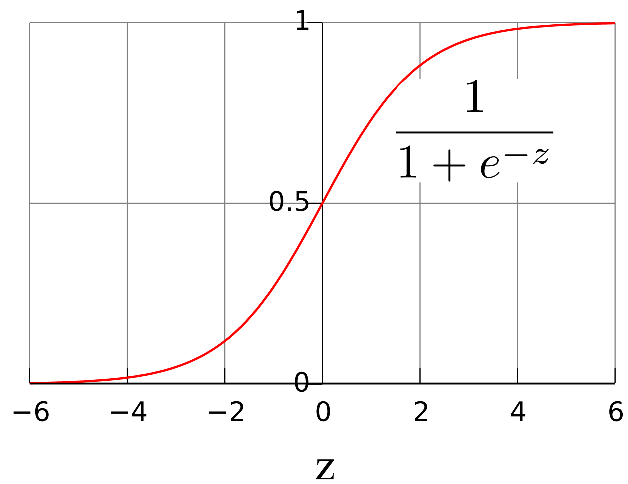
(Figure from tutorial by Sean Borman)

What you should know

- Mixture of Gaussians
- EM for mixture of Gaussians:
  - Coordinate ascent, just like k-means
  - How to "learn" maximum likelihood parameters (locally max. like.) in the case of unlabeled data
  - Relation to K-means
    - Hard / soft clustering
    - Probabilistic model
- Remember, E.M. can get stuck in local minima,
  - And empirically it ***DOES***

Logistic Regression

Logistic function (Sigmoid):



- Learn P(Y|**X**) directly!
  - Assume a particular functional form
  - Sigmoid applied to a linear function of the data:

$$P(Y=1|X) = \frac{1}{1+\exp(w_0+\sum_{i=1}^{n} w_i X_i)}$$

$$P(Y=0|X) = \frac{\exp(w_0+\sum_{i=1}^{n} w_i X_i)}{1+\exp(w_0+\sum_{i=1}^{n} w_i X_i)}$$

Features can be discrete or continuous!

Naïve Bayes *vs.* Logistic Regression

Learning: h:**X** $\mapsto$ Y

X – features

Y – target classes

Generative
- Assume functional form for
  - P(X|Y) **assume cond indep**
  - P(Y)
  - Est. params from train data
- Gaussian NB for cont. features
- Bayes rule to calc. P(Y|X= x):
  - P(Y | **X**) $\propto$ P(**X** | Y) P(Y)
- **Indirect** computation
  - Can generate a sample of the data
  - **Can easily handle missing data**

Discriminative
- Assume functional form for
  - P(Y|X) **no assumptions**
  - Est params from training data
- Handles discrete & cont features
- Directly calculate P(Y|X=x)
  - Can't generate data sample

Naïve Bayes vs. Logistic Regression

[Ng & Jordan, 2002]

- Generative vs. Discriminative classifiers
- Asymptotic comparison

  (# training examples → infinity)

  – when model correct
    - NB, Linear Discriminant Analysis (with class independent variances), and Logistic Regression produce identical classifiers

  – when model incorrect
    - LR is less biased – does not assume conditional independence
      - **therefore LR expected to outperform NB**

Naïve Bayes vs. Logistic Regression

[Ng & Jordan, 2002]

- Generative vs. Discriminative classifiers
- Non-asymptotic analysis
  - convergence rate of parameter estimates,
    (n = # of attributes in X)
    - Size of training data to get close to infinite data solution
    - Naïve Bayes needs $O(\log n)$ samples
    - Logistic Regression needs $O(n)$ samples

  - Naïve Bayes converges more quickly to its ***(perhaps less helpful)*** asymptotic estimates

Some experiments from UCI data sets

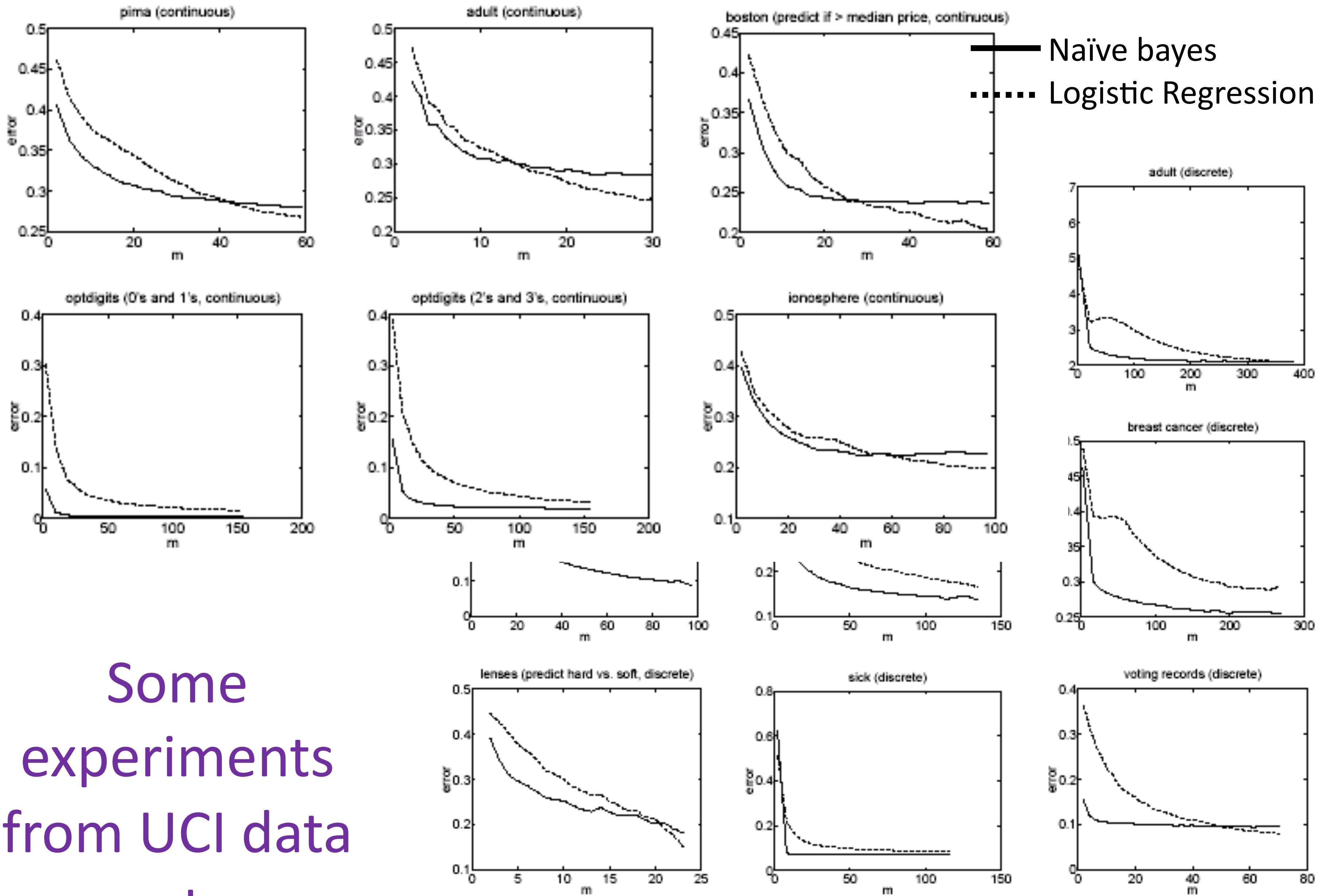


Figure 1: Results of 15 experiments on datasets from the UCI Machine Learning repository. Plots are of generalization error vs. m (averaged over 1000 random train/test splits). Dashed line is logistic regression; solid line is naive Bayes.

Logistic regression for discrete classification

Logistic regression in more general case, where set of possible *Y* is $\{y_1,\ldots,y_R\}$

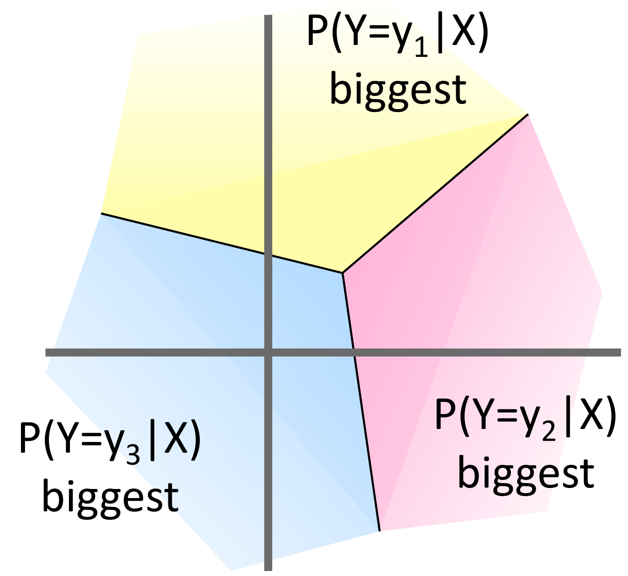
- Define a weight vector w_i for each y_i, i=1,…,R-1

$$P(Y=1|X) \propto \exp(w_{10} + \sum_i w_{1i} X_i)$$

$$P(Y=2|X) \propto \exp(w_{20} + \sum_i w_{2i} X_i)$$

…

$$P(Y=r|X) = 1 - \sum_{j=1}^{r-1} P(Y=j|X)$$



Logistic regression for discrete classification

- Logistic regression in more general case, where Y is in the set $\{y_1,\ldots,y_R\}$

for $k<R$

$$P(Y=y_k|X) = \frac{\exp(w_{k0} + \sum_{i=1}^{n} w_{ki}X_i)}{1 + \sum_{j=1}^{R-1} \exp(w_{j0} + \sum_{i=1}^{n} w_{ji}X_i)}$$

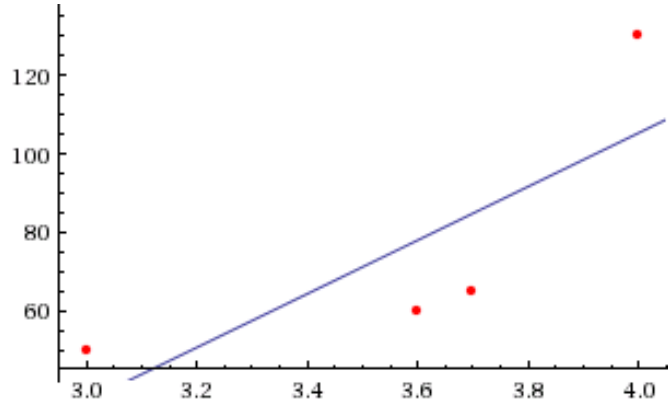
for $k=R$ (normalization, so no weights for this class)

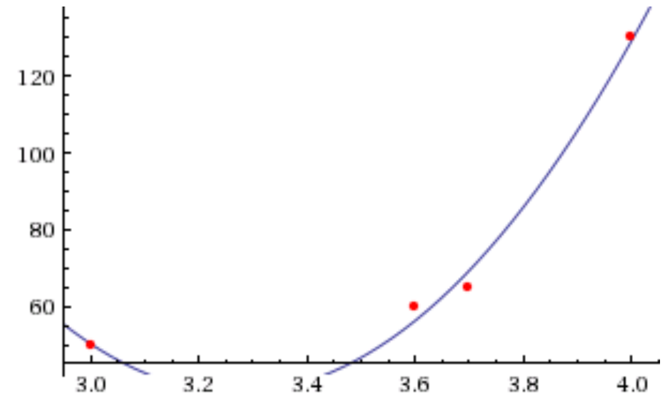
$$P(Y=y_R|X) = \frac{1}{1 + \sum_{j=1}^{R-1} \exp(w_{j0} + \sum_{i=1}^{n} w_{ji}X_i)}$$

Features can be discrete or continuous!

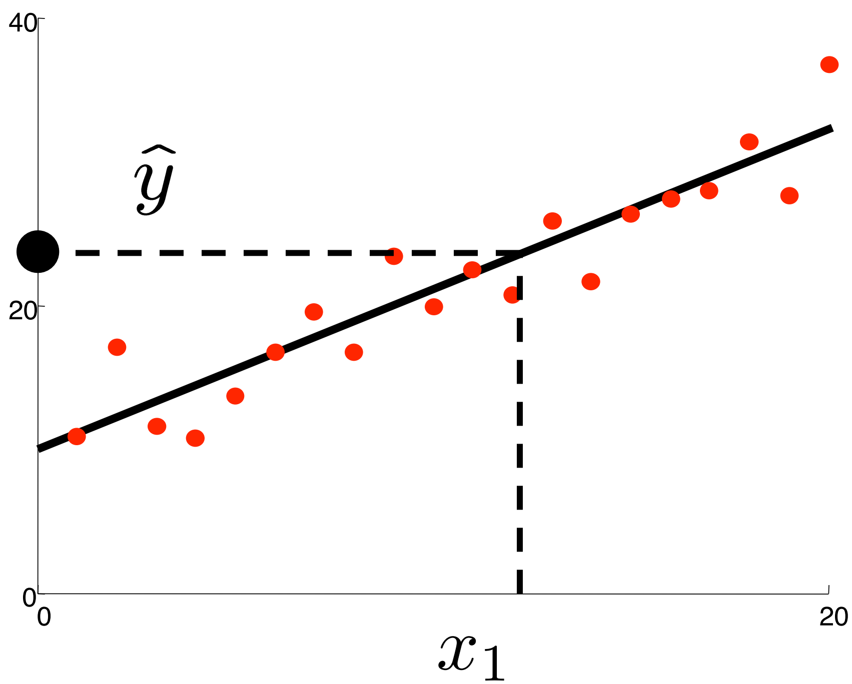
Prediction of continuous variables

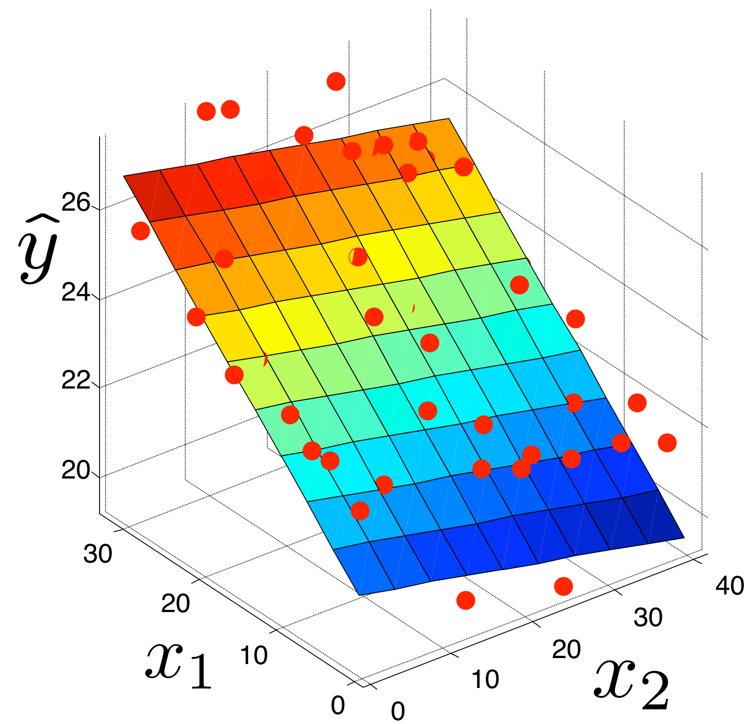
- Billionaire says: Wait, that's not what I meant!
- You say: Chill out, dude.
- He says: I want to predict a continuous variable for continuous inputs: I want to predict salaries from GPA.
- You say: **I can regress that…**





Linear Regression





Prediction

$$\hat{y} = w_0 + w_1 x_1$$

Prediction

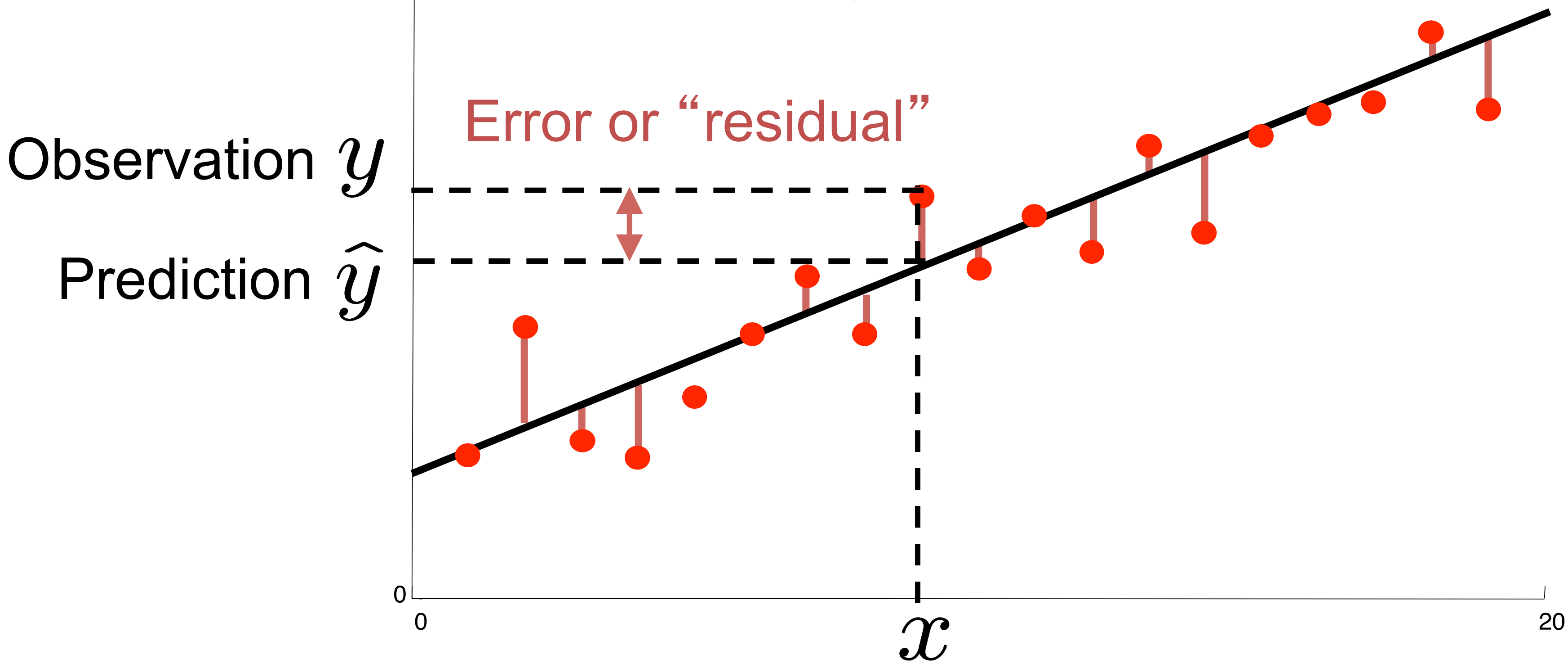
$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2$$

Ordinary Least Squares (OLS)

$$\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i \left(y_i - \sum_k w_k x_k^{(i)} \right)^2$$

sum over data points

features



The regression problem

- Precisely, minimize the residual squared error:

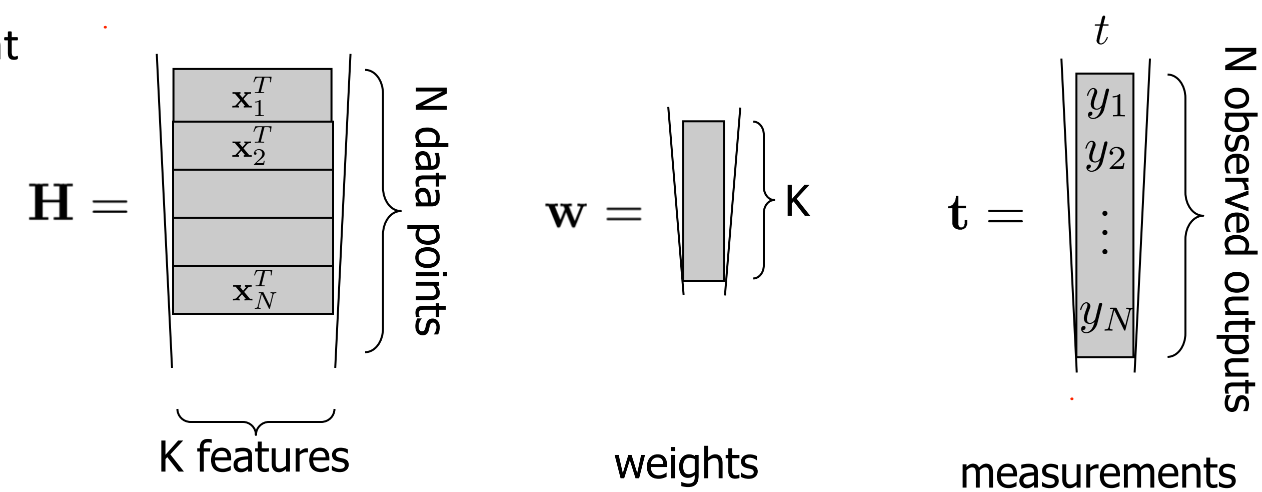
$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_i \left(y_i - \sum_k w_k x_i^k \right)^2$$

Regression: matrix notation

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_i \left(y_i - \sum_k w_k x_i^k \right)^2 = \sum_i \left(\mathbf{x}_i^T \mathbf{w} - y_i \right)^2$$

$$\mathbf{w}^* \quad = \quad \arg\min_{\mathbf{w}} \underbrace{(\mathbf{H}\mathbf{w} - \mathbf{t})^T(\mathbf{H}\mathbf{w} - \mathbf{t})}_{\text{residual error}}$$

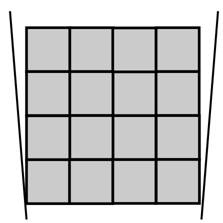
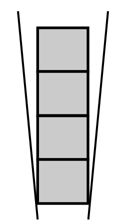
One data point per row



Regression solution: simple matrix math

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \underbrace{(\mathbf{H}\mathbf{w} - \mathbf{t})^T(\mathbf{H}\mathbf{w} - \mathbf{t})}_{\text{residual error}}$$

$$\text{solution: } \mathbf{w}^* = \underbrace{(\mathbf{H}^{\mathrm{T}}\mathbf{H})^{-1}}_{\mathbf{A}^{-1}} \underbrace{\mathbf{H}^{\mathrm{T}}\mathbf{t}}_{\mathbf{b}} = \mathbf{A}^{-1}\mathbf{b}$$

where $\mathbf{A} = \mathbf{H}^{\mathrm{T}}\mathbf{H} =$  $\mathbf{b} = \mathbf{H}^{\mathrm{T}}\mathbf{t} =$ 

K×K matrix of feature correlations

K×1 vector

But, why?

- Billionaire (again) says: Why sum squared error???
- You say: Gaussians, Dr. Gateson, Gaussians…
- Model: prediction is **deterministic** linear function **plus Gaussian noise**:

$$y_{\text{observed}} = \sum_k w_k x_k + \epsilon \qquad\qquad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

- Learn **w** using MLE:

$$\Pr(y_{\text{observed}} \mid \mathbf{x}, \mathbf{w}, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(y_{\text{observed}} - \sum_k w_k x_k)^2}{2\sigma^2}}$$

Maximizing log-likelihood

Maximize wrt w:

$$\ln P(\mathcal{D} \mid \mathbf{w}, \sigma) = \ln \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^N \prod_{j=1}^{N} e^{\frac{-[t_j - \sum_i w_i h_i(\mathbf{x}_j)]^2}{2\sigma^2}}$$

$$\arg\max_w \ln \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^N + \sum_{j=1}^{N} \frac{-[t_j - \sum_i w_i h_i(x_j)]^2}{2\sigma^2}$$

$$= \arg\max_w \sum_{j=1}^{N} \frac{-[t_j - \sum_i w_i h_i(x_j)]^2}{2\sigma^2}$$

$$= \arg\min_w \sum_{j=1}^{N} [t_j - \sum_i w_i h_i(x_j)]^2$$

(note that the notation here is slightly different)

Least-squares Linear Regression is MLE for Gaussian noise!!!